Autonomous Robots 6, 247–264 (1999)

Component Specifications for Robotics Integration

E. MESSINA, J. HORST, T. KRAMER, H. HUANG AND J. MICHALOSKI
National Institute of Standards and Technology, Gaithersburg, MD 20899

Abstract. Robotics researchers have been unable to capitalize easily on existing software components to speed up their development efforts and maximize their system's capabilities. A component-based approach for building the software for robotics systems can provide reuse and sharing abilities to the research community. The software engineering community has been studying reuse techniques for three decades. We present several results from those efforts that are applicable to the robotics software integration problem. We describe how to specify a software component so that a potential user may understand its capabilities and facilitate its application to his or her system. At the National Institute of Standards and Technology, we have developed a three-stage, component-specification approach. We illustrate this approach for a component that is relevant to robotics.

Keywords: software components, software reuse, frameworks, intelligent systems, software architectures

1. Robotic Research and Software Reuse: Two Parallel Efforts

Robotics research has long been at an impasse: unless appropriate technologies emerge, ineffective and inefficient sharing of software will continue to impede the goal of truly intelligent, robust, robotic system behavior. This paper discusses the use of component specifications to achieve sharing and integration of robotic software modules. We begin by providing background on component specification and reuse activities in the general software engineering community. A brief discussion of some efforts in the robotics domain that are relevant to component-based development follows. Finally, an approach to component specifications developed and applied at the National Institute of Standards and Technology provides a concrete instance. Software reuse—using existing software artifacts during the construction of a new system—has long been a goal of the software engineering community (Nauer and Randel, 1968). Numerous efforts have been underway in academia and industry that address the challenges of creating software components so that they are suitable for reuse. Research issues include specification, implementation, retrieval, and assembly of components. The robotics research community may leverage certain aspects of these efforts in order to communicate their particular requirements and the functionality of those components. A common understanding of the application domain, vocabulary, and architectures (Chen and Cheng, 1997) contributes to a concrete framework benefiting reuse. Krueger (1992) states that the few success stories in reuse occur in similar environments having a common, domain-specific frame of reference. Being able to leverage existing software components would reduce development efforts for researchers. Instead of having to write all of the software for its robot testbed, a research team could concentrate on the software that is unique to their area of interest. Furthermore, building systems using proven "best of class" components developed elsewhere should lead to more capable robotic implementations overall.

A main research area for software reuse has been component specification. Despite the progress made in this area, there are several important obstacles to the application of these results and approaches directly to the integration of robotic research: the bulk

Certain commercial products or company names are identified in this paper to describe our study adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products or names identified are necessarily the best available for the purpose.

of the commercial efforts in specifications for reuse have been targeted towards business software development, which addresses a different set of requirements than does development of real-time software that interacts with and controls hardware components. Several of the current efforts rely on fairly constrained definitions of environments in which reuse is possible. They assume that the reuse will occur within a particular development environment (Kara, 1997). It is not realistic to assume that robotics research institutions will abandon their current development environments in favor of a standardized one, even if that would provide reuse possibilities.

At the National Institute of Standards and Technology, we have been developing the specifications of software components for intelligent control, a domain that includes robotics. We define intelligent control as the capability for a complex system to successfully perform complex physical tasks in the presence of uncertainty and unpredictability. In considering the component-based approach to software design and development, we initially found inspiration in the design paradigm prevalent for certain hardware domains. In hardware design, a high-level functional decomposition is generated, along with requirement specifications for individual components. Designers use hardware catalogs to locate candidate components that match their specifications. In certain hardware domains, such as printed circuit board design, integrated design and simulation tools allow the designer to evaluate the overall system performance using the candidate chips. Specifications that define the characteristics and behavior of a chip exist in languages such as the Very High Speed Integrated Circuits Hardware Description Language (VHDL) (IEEE, 1994). Eventually, we expect to see an analogous capability emerge for software, allowing designers to locate available components that match their criteria and evaluate their performance through simulation that is derived from the specification itself. In the nearer term, we seek to enable software designers to browse catalogs of available components that are described in a meaningful way. Our approach is unique in that it takes a multi-stage view of software component specifications. There are three potential levels of specification for a software component:

— a specification template which identifies relevant aspects for a component;
— specific, natural language text instantiations that fill in the categories in the generic class;
— formal versions of the specific instantiations.

Much of the literature on component specification and reuse is targeted towards domain-specific algorithms that are not based on engineering, scientific, or real-time applications. We were interested in evaluating the issues in specifying components in those domains where potential users of a component often want to compare their system requirements with semantic characteristics of multiple competing components. For example, a user may need to know whether an algorithm will not converge given certain input conditions. Furthermore, the target-system's performance characteristics are often not fully known, so that testing and simulation are especially critical in the context of a prototype system.

We began our research by identifying a class of algorithms that would lend themselves to reuse in industrial or research settings, developing specifications for that class of algorithms, and studying how to represent the specifications so that they can be used in several ways. To illustrate the general concepts about software components that are presented, we include a description of our results thus far in generating and validating component specifications for a class of control software components.

2. Background and Relevant Efforts

2.1. Academic Work

There are many definitions of the term "component". When the inner workings of a component are not open for inspection, it is generally referred to as a black box component (Short, 1997; Szyperski, 1998). Conversely, a white box component's insides (typically source code) are available for customization, extension, or other modification. We shall use the term component to mean a unit of software with clearly defined interface and functionality. This definition allows components to be white box, but does not require it. The majority of component-based software research is attempting to address the reuse challenge. In order to facilitate successful reuse, an abstraction process must take place that reduces the level of details that are unimportant to the system developer (Krueger, 1992). The creation of meaningful summaries of component behavior in the form of component specifications is just such an abstraction process.

There are three main areas of characterization required to define a software component: its static aspects, its dynamic characteristics, and its semantic content. Most descriptions of software components for

reuse define the *signature* of a piece of software. The signature primarily describes the static aspects of code, such as the call interface, data flows, and operating system requirements. Several efforts have been aimed at detailing the dynamic aspects of a piece of software, that is, its run-time characteristics. For example, the C++ Standard Template Library includes execution time bounds in the specification of functions (Musser and Saini, 1996). Another approach using run-time information is based on historical traces. Histories of permissible state transitions have been used as part of a component's specifications (Liskov and Wing, 1993). Capturing the semantic aspects, i.e., the real meaning and function of a piece of software, poses the most challenges. A semantic description of a component is a form of information abstraction. Krueger (1992) refers to the process of abstraction as an attempt to reduce the cognitive distance, which he defines as "the amount of intellectual effort that must be expended by software developers in order to take a software system from one stage of development to another." Therefore, the challenge is not only to capture the semantics of a component, but also to do so at an appropriate level or levels of abstraction. Carefully chosen and appropriately utilized component specifications can meet such a challenge.

Several component specification approaches attempt to create a language or taxonomy for describing components such that they can be matched with descriptions of required functionality or features. Zaremski and Wing (1996) define specification matching as the "process of determining if two software components are related." Specification matching can facilitate retrieval based on syntax and semantics, adaptation for reuse, and substitution of one component for another. Kazman et al. (1997) describe the use of architectural elements for classification matching. They categorize the run-time characteristics (such as, times of control acceptance, times of data acceptance, and state retention) and the static features (such as, whether the data and control scopes are virtual or physical and whether the component binds at invocation or execution) of a component and match those against specified criteria. The full semantic description of a component can be challenging to produce, especially if trying to understand a piece of software written by someone else. This has led to attempts to extract component specifications from code automatically or semi-automatically. Approaches vary widely. For example, Basili and Abd-El-Hafiz (1992) describe the Computer Aided Reuse Engineering Functional Specification Qualifier, which is a knowledge based approach that finds the abstract specifications of a program's loops and produces specifications in a first order predicate language. Podgurski and Pierce (1993) retrieve software components by modeling the inputs and desired outputs and statistically exercising a set of candidates with random inputs to find the ones whose outputs most closely match the desired ones.

Component specifications may, in some cases, be more readily derived by taking a domain-specific approach (Batory and O'Malley, 1992; Demeyer et al., 1997; Nierstrasz et al., 1992; Nierstrasz and Meijler, 1995). The universe of possible options and the problems addressed is bounded within domain-specific approaches. Domain-specific component reuse is related to framework reuse. A framework is a collection of components with defined relationships between them (Pree, 1997). Frameworks may be designed to have designated components that are replaceable or customizable. Some of the literature refers to these designated variable components as hot spots (Pree, 1997) or axes of variability (Demeyer et al., 1997).

We briefly describe some examples that are especially relevant to robotics researchers. The Reconfigurable Modular Manipulator System (RMMS) (Paredis et al., 1997; Stewart et al., 1997) at Carnegie Mellon University provides an environment that supports assembly of robotic manipulators through the use of hardware and software building blocks. The system is a framework of port-based objects (PBOs) built upon a custom real-time operating system. PBOs are based on the port-automaton theory (Steenstrup et al., 1983). The PBOs are independent processes that communicate with other PBOs via input and output ports. They interact with sensors and actuators via resource ports. Configuration constants are used to tailor generic components for specific hardware or applications. PBOs are not defined semantically in a formal manner. A text configuration file describes a port-based object. The configuration file contains the module name, its natural language description, names of input and output variables (ports), input and output configuration constants, whether the task is periodic or aperiodic, and process execution frequency. Module-specific local configuration parameters, such as gains, may also be included in the configuration file. Selecting PBOs and connecting their corresponding inputs and outputs configures a system. A graphical tool is available to support software composition using PBOs. There does not appear to be a means of retrieving components matching particular specifications.

Another example from the robotics domain is the Interactive Benchmark for Planning Algorithms on the Web (Piccinocchi et al., 1997). This web-based environment provides users the ability to compare path planning algorithms for car-like vehicles. Using a client-server model, users can exercise several planning algorithms already implemented within the environment by either using available problem sets, or defining their own. A problem is posed by selecting one of five vehicles, defining polygonal obstacles and start and goal positions for the vehicle. The solution is presented graphically as the path found, along with data such as path length and execution time. Users can connect their own planners to the environment. This aspect requires a component specification for the planner algorithm. The overall environment can be considered a domain-specific framework providing an infrastructure (HTML (Berners-Lee and Connolly, 1993), CGI scripts (McCool, 1994), and TCP/IP (Postel, 1981)) for one variable component. Because the environment is tailored for a very restricted domain, the specification for the planner component describes simple text data input and output files. The input file lists the number of obstacles and a sequence of vertices for each obstacle. The output file contains a path defined by a sequence of linear, circular, or elliptical segments. There is no explicit semantic representation used in this testbed and no performance data are included.

2.2. *Specifications in Industry*

Commercial efforts have become much more visible and viable in the past few years. In the book by Szyperski (1998), the focus is on reusable assets and typically refers to blackbox reuse. His definition of components is "binary units of independent production, acquisition, and deployment that interact to form a functioning system." Most of the leading efforts, such as Microsoft (COM, OLE, ActiveX, COM+) (Brockschmidt, 1995; Chappell, 1996; Denning, 1997), Sun Microsystems (Java, Java Beans) (JavaSoft, 1998), and Object Management Group (OMA, CORBA) (OMG, 1996, 1997), are at the "wiring" level. They concentrate on the communication infrastructure and gluing together of components (Pree, 1997; Szyperski, 1998).

Component specifications are of increasing importance to the manufacturing industry as it is in the midst of a major shift in technology from closed, proprietary systems to the era of open, plug-and-play systems. The leading plug-and-play standardization efforts all define an open-architecture and component interface specification, but differ in their approaches. The Open System Environment for Controller (OSEC) working group, founded by six Japanese companies, defines the Factory Automation Equipment Description Language (FADL) based on the PERL object-oriented programming language (SML Corporation, 1998). With a more powerful programming language like FADL, reusability would result from standardized libraries available for common applications. The Open System Architecture for Controls within Automation Systems (OSACA) started as a joint European project in May 1992 but has broadened the scope to define a worldwide, vendor-neutral system architecture containing a communication specification, a reference architecture and a configuration specification (OSACA, 1998). Reusability is based on conformance to the reference architecture, which describes external interfaces for a set of standard modules. The Open Modular Architecture Controller (OMAC) (Bailo et al., 1994) Application Programming Interface (API) (OMAC API Workgroup, 1997) workgroup has developed an object-oriented framework that includes a control class hierarchy, plug-and-play modules aggregated from the class hierarchy, and a model of collaboration. Reusability is based on the framework model (Fayad and Schmidt, 1997).

3. Software Specifications for Intelligent Control Components

We have been researching methods to design software component specifications that are standardizable and comprehensive and would provide the basis for component repositories usable for intelligent control. Our research seeks to improve the efficiency of communicating a new algorithm's capabilities and characteristics, so that potential users can evaluate it. We use the terms algorithm and component interchangeably in this section since the focus of our work is to facilitate distribution and leveraging of algorithms, not necessarily the comparison of different implementations of the same algorithm. Since the information is crucial to the design of a system, some of the specification categories do require data about the algorithm as implemented in a component.

The research described herein targets a particular set of aspects pertinent to component specifications. The domain is focused on intelligent control. The scope

of the specification categories goes beyond just the "wiring" and signature aspects of a component. We include semantics and certain implementation characteristics so as to allow system developers to effectively communicate and comprehend a component's capabilities and applicability to their system. We are investigating methods to use specifications in design and simulation tools during the system design process.

Taking a domain-specific approach to component analysis, we targeted a particular family of algorithms relevant to intelligent control, namely pose estimation. This enabled us to extract the significant semantic information without having to be exceedingly broad. We have taken a three-part approach to component specification. Initially, generic specification categories to be filled in for each algorithm are defined. The generic categories are a template to be used to define the operation and performance of a particular component in the class. Secondly, for a particular algorithm or component, the developer fills in each slot in the template. This creates the particular, instantiated specification in a natural language format. The final step in specification is to convert the natural language descriptions into a formal language. The multi-stage development of specifications within a domain-specific context provides a robust approach to component definition.

There are several advantages to considering the formal representation of an algorithm. Formal languages provide an unambiguous specification of information about the component. They also hold the potential for supporting simulation of the component's execution to verify how it will fit into the overall system being assembled. Ultimately, if automated component retrieval, matching, and composition is desired, some form of formal language will be required. Formal methods have not been widely deployed outside of specialized areas due to the lack of familiarity with the notations and the difficulties of using them (Baresi et al., 1997). Nevertheless, we believe that their advantages outweigh the challenges of adopting them. Furthermore, we anticipate that, in the long run, tools will be able to extract the formal expressions, translate between representations, and utilize them without exposing the developers or users of the components to the formal representations.

The algorithm class that we have focussed our research on performs position and orientation estimation for a physical object using visual sensing (also referred to as "part pose estimation"). This is a class for which several algorithms have been published, enabling us to examine the breadth of specification categories applicable to these components. This class provides a sufficiently rich problem set, involving several aspects that are relevant to developers of controllers for robotics and other advanced automation applications. Key elements include sensory inputs, mathematical computation, access to an external database (such as a Computer-Aided Design model of the part), spatial reasoning, and a range of algorithm options, such as pose matching two-dimensional information against two-dimensional or three-dimensional data. Available resources in-house enabled us to validate the specifications through implementations.

We validated the specifications for a component within an inspection system that was implemented based on the NIST Real-Time Control System (RCS) architecture (Albus, 1996, 1997). The RCS architecture has been used in development of several robotic applications (see Herman and Albus, 1988; Murphy et al., 1988, 1993; Stouffer and Russell, 1995 as examples). Although the particular testbed for which the part pose estimation component was developed is not strictly speaking a robotic one, the problem being solved is very compatible with the type encountered by robot developers. Essentially, the system has to extract information about its environment through the use of sensors and update an internal model of the world using a priori knowledge in order to generate a plan of action that reflects its goals and the state of the environment. The images captured from the cameras inform the system about its environment. The system has a goal to inspect a part's dimensions. Dimensional inspection involves measuring the relative geometry of surface features and determining whether they are within tolerance. Examples of feature geometry evaluated include shapes of smooth surfaces, distances between edges, positions of holes, and diameters and shapes of holes. The system has a priori knowledge or expectations about the datum geometry of the part and its location. Comparing the expected world model with the sensed world model, the system alters its inspection plan based on the actual position of the part.

The inspection system testbed provides an RCS-based framework with the part pose estimation subsystem designated as a hot spot. From a framework perspective (Huang, 1996), RCS prescribes a multi-dimensional decomposition of a system, built upon generic controller nodes. The nodes are assigned specific responsibilities at different levels of abstraction determined by their functionality in one axis, and

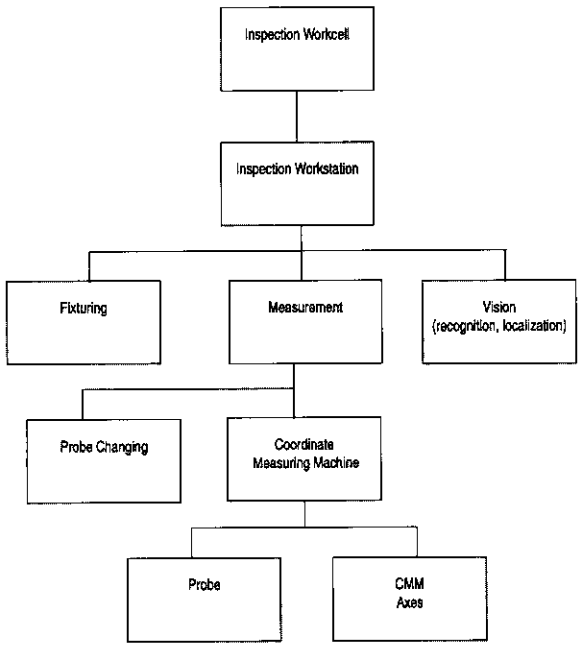


Figure 1. RCS hierarchy for an inspection testbed.

their response time in another. The varying levels of abstraction help reduce the cognitive distance for engineers during the design process. This allows them to focus on one level of abstraction at a time and on the relevant variables and parameters for that level only. Communication pathways and component responsibilities within the hierarchy and between nodes are defined within the RCS framework. All controller nodes contain behavior generation (BG), sensory perception (SP), value judgement (VJ), and world modeling (WM) functionality. The part pose estimation component corresponds to sensory processing and world modeling functions within the Vision node in the simplified hierarchy diagram in Fig. 1.

3.1. *Step 1: Generic Categories for the Specifications*

The generic categories for component semantics that were defined can be thought of as a set of questions that a potential user of the component may ask its developer in order to ascertain whether the component is appropriate for use in a new system. The questions should address the static, dynamic, and semantic aspects of the software component. The categories that were selected to describe the part pose estimation component are listed below.

Problem Definition: What problem is this component intended to solve? What are the potential and known application areas for this component? For each application area, give a set of competing components with references.

Input Data: What is the input data to the component? What input data sets were used in the testing? What are the dimensional units of the input data? What is the format of the input data? Are the representations chosen for the input data consistent with the expected or typical upstream components? What are the input parameters required (if any) and how do they affect the operation and performance of the component? Input parameters are the design values (typically static) that affect the operation of the algorithm, whereas input data values are the input from the world, e.g., a sensor, that are to be transformed in some way. Are there any input parameters that allow the user to specify the type and/or format of the output?

Output Data: What are the outputs of the component? What is the format of the output data? How are the various formats for the output data specified? For instance, if the output data is contained in files, what are the file formats? Are the representations chosen for the output data consistent with the expected or typical downstream components?

Transfer and Feedback Relations: How do the inputs relate to the outputs, i.e., what are the transfer and the feedback relations? If one can describe these relations analytically, describe these equations, e.g., are they linear or non-linear, and state these equations, e.g., define meanings of variables and write out all relations. Under what conditions are the equations over-determined or under-determined?

Input Data Constraints: What are the constraints on the format and nature of the input? There may be constraints on the nature of the input beyond what is inherent in the nature of the component. For example, the number of elements in an array might be constrained to be even, or the difference between input numbers might be constrained to be greater than some value, or the number of input values might be constrained to be in some range.

Environmental Constraints: 'Environment' means factors external to the computing system and its data which affect the performance of the component. If the task of the component is to examine or manipulate physical objects, what are the constraints on the format and nature of those objects? For instance,

what are the rigidity, size, shape, color, surface finish, or illumination conditions? If there is a sensing device, what are the constraints on the type and use of the sensing (e.g., structured light, CCD camera, range camera)? Does the object need to be placed in some approximate pose? Are there special configurations of the environment that might cause the component to fail?

Knowledge Data Constraints: What are the constraints on the format and nature of the knowledge data? For example, can a CAD drawing in some standard format be used for matching sensed features to model features?

Computing Constraint: What, if any, are the operating system requirements of the component? In what computer language is the source code written? Is the source code available? Are there any system architecture requirements for using the component? What kind of computing hardware is required by the component? How much RAM memory and disk space is needed? Are there any constraints on the numerical precision of the processing system?

Internal Data Representation: If there is some knowledge (i.e., not input/output data) that is explicitly represented within the component, what is the format of the representation?

Speed: Based on actual examples run on specific computers, how fast does the component run? What is the execution time of each of the subcomponents of the component? If speed depends on the size or type of input data, give the speed of execution for a fixed and standard size or type of input.

Complexity: What are the relations defining computational complexity of the component (there may be more than one such relation) as functions of the input variables? Complexity relates closely to speed. But an analysis of complexity typically does not deal with the initial fixed cost of the component or with the size of the constants by which various terms of the complexity must be multiplied. What assumptions are made in the complexity analysis?

Benchmarks: If there is a standard test suite (a set of benchmarks) for components performing the same task, what are the benchmarks and how does the component perform against them? Are there optimal components that can produce the ideal output? How is optimality defined for each component? Are there other measures of performance, e.g., statistical measures? If so, how does the component perform based on these measures?

Robustness: How robust is the component, i.e., how does the component perform in the presence of large perturbations on a subset of its data values? Such perturbations have been called replacement noise, e.g., when object recognition is the task and feature recognition is a subtask and a feature of Type A is thought to be of Type B by the algorithm. Replacement noise is usually large on a small subset of the data and probabilistic noise is small on a large subset of the data. How does the component perform as replacement noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? Several aspects of this type of noise can vary, for instance, the size and type of the perturbations and the size of the subset of all data values affected. Is the component able to perform well (or at all) if the input is outside of the specified region? State all models that exist for replacement noise. For instance, what is the model for the 'ideal' world? What is the model for large perturbations (e.g., mismatch) on this ideal world? What is the criterion function for measuring the difference between noisy output and ideal output (Haralick et al., 1989)?

Noise: How does the component perform in the presence of small perturbations, i.e., probabilistic noise, on all data values? How does the component perform as this type of noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? Several aspects of this type of noise can vary. For instance, both the size and type of the perturbations can vary. State all models that exist for this type of noise. For instance, what is the model for the 'ideal' world, what is the model for small perturbations on this ideal world, and what is the criterion function for measuring the difference between noisy output and ideal output (Haralick et al., 1989)?

Convergence: Is the component iterative or closed form? If iterative, under what (if any) conditions is convergence guaranteed? Does the component converge to the global solution?

Errors: What is the error criterion, e.g., least squares? What kinds of input errors and/or internal errors does the component detect? What does it do if such errors are detected? Are there error recovery procedures?

Reliability: If the algorithm is available as source code, a library, or embedded in a system, how reliable is it? How reliable is the component? Are there any known bugs? What reliability tests has the component passed, such as the checking provided by commercially available reliability tools,

or a theoretical correctness proof? Example entities checked by such tools are uninitialized local variables, uninitialized malloc'd memory, using freed memory, overwriting array bounds, over-reading array bounds, memory leaks, file descriptor leaks, stack overflow errors, and stack frame boundary errors. Has there been testing by some sort of coverage test tool, which keeps track of which code is executed in a given session? Coverage tools give a sense of how much of the code was exercised by other reliability tests. The granularity can be at the function, block, or line level. A coverage tool will give the potential user greater confidence in a software component's reliability if it had been covered 100% during testing and errors were cleaned out of it.

Testing and Analysis: What experiments have been done with the component? If input data was varied over a set of variables, what criteria were chosen to sample the space of variables in order to generate sample input data? Was Monte Carlo testing done? Are there simulators available that generate input data? What kinds of analysis have been done on the results? What kinds of graphs and tables have been produced and what is their format? What statistical methods have been used?

Application Experience: How widely has the algorithm or component been used? What is the reported experience? What are the potential application areas for this algorithm and, for each application area, give a set of competing algorithms with references?

Upstream and Downstream Requirements: What kinds of components or algorithms are typically executed prior to or subsequent to this algorithm? Are the representations chosen for the input and output data consistent with the expected or typical upstream and downstream components? Does anything in the component constrain the upstream or downstream components that must be used? The types of upstream and downstream components may depend on the particular area to which the component is applied.

Parallelizability: Can the component be parallelized? Has it been? How does parallelizing affect performance?

Modularizability: Can subcomponents of the component be modularized? Have they been? How might modularization affect component performance?

Nature of Interaction: How is the component used? Can it be used by function call, or is the component part of a system that runs in client-server mode, or some other more complicated mode?

Coding Style: How is the code written, e.g., in functional, procedural, recursive, or object-oriented style?

Compliance to Standards: Which (if any) standards (published or de facto) are used and complied with?

3.2. Step 2: Natural Language Instantiation of the Specification

Given a template to fill in for a particular algorithm class, the developer can fill out all the slots with available information. It is possible that not all of the data will be known or be applicable. The natural language description of the algorithm, generated using the generic template as a guide, can be used directly to communicate a component's capabilities, limitations, and requirements. If component natural language descriptions are made publicly available in text or other format, readily available tools can be applied to locate and assess the components. A search engine can be used to locate candidate components based on certain keywords. Potential users can also browse through the descriptions as they would through a hardware components catalog.

The above component template was filled in for published algorithms as well as for an in-house developed algorithm. We include the instantiation for a part pose algorithm (Tan et al., 1992, 1994, 1996) to illustrate what the natural language specifications look like. The algorithm computes an estimate of the position and orientation (pose) of any object where the inputs to the algorithm are the matched model and sensed feature pairs. The algorithm consists of noniterative, closed form expressions, which is rare for 3D pose estimation algorithms. This is accomplished by assuming that the ground plane on which the object lies is known, which is a common assumption for many applications.

Problem Definition: What problem is this component intended to solve? This component computes the location and orientation of a laminar part. What are the potential application areas for this algorithm and, for each application area, give a set of competing algorithms with references? Pose estimation of automobiles on roads of known orientation and rigid objects on flat surfaces of known orientation. Some competing algorithms are (Dementhon and Davis, 1995; Haralick, 1992; Huttenlocher and Ullman, 1990).

Input Data: What is the input data to the algorithm? An array of N vectors of the following nine real numbers (doubles) for each $i = 1, 2, \ldots, N$, $(a_i, b_i, c_i, x_i^0, y_i^0, z_i^0, \alpha_i, \beta_i, \gamma_i)$, where N is the number of lines in the image that corresponds to lines in the model, (a_i, b_i, c_i) is the unit vector of parameters that solves the equation, $a_i u + b_i v + c_i = 0$ for all on the ith measured line in the image, (x_i^0, y_i^0, z_i^0) is the position vector of the initial point of the corresponding model line segment in model coordinates, and $(\alpha_i, \beta_i, \gamma_i)$ is the unit directional vector of the same model line segment also in the model coordinates. Also required for input are the parameters for perspective transformation $(t_x, t_y, t_z, \delta, \varepsilon, \zeta, f)$, where (t_x, t_y, t_z), is the vector of translation from the origin of the machine coordinate system to the center of the camera lens, $(\delta, \varepsilon, \zeta)$ is the vector specifying the roll, pitch, and yaw of the camera in radians (yaw is a counter-clockwise spin around the z-axis, pitch is a counter-clockwise spin around the x-axis, and roll is a counter-clockwise spin around the y-axis), and f is the camera focal length. What input data sets were used in the testing? Tan has models of automobiles. The authors have a slightly modified cube for testing which is defined in Mathematica. What are the dimensional units of the input data? (a_i, b_i, c_i) are dimensionless, (x_i^0, y_i^0, z_i^0) are in meters, and $(\alpha_i, \beta_i, \gamma_i)$ are in radians. (t_x, t_y, t_z) are in meters, $(\delta, \varepsilon, \zeta)$ are in radians, and f is in meters.

What is the format of the input data? All input arrays are arrays of doubles. Are the representations chosen for the input data consistent with the expected or typical upstream algorithms? The expected upstream algorithm is an algorithm matching a sensed feature to a model feature and, since the Tan algorithm requires input lines to match edges on a planar polygonal object, the matching algorithm must produce line matches as well. However, if point or line segment matches are all that are available from the matching algorithm, line parameters can easily be generated from them.

What are the input parameters required (if any) and what meaning do they have for the operation and performance of the algorithm? None required. Are there any input parameters that allow the user to specify the type and/or format of the output? No.

Output Data: What are the outputs of the algorithm and what is the format of the data? An array of four doubles, (x, y, θ, k), where (x, y, θ) represent the two dimensional position and orientation of the part and k represents the scale of the part (in case the part measured is a scaled version of the model). The remaining three parameters, (z, θ, Ψ), required to fully specify the three dimensional position and orientation, are assumed to be known and equal to zero a priori. How are the various formats for the output data specified, for instance, if the output data is contained in files, what are the file formats? An ANSI C-compliant array.

What is the format of the output data? All output arrays are arrays of doubles. Are the representations chosen for the output data consistent with the expected or typical downstream algorithms? Yes.

Transfer and Feedback Relations: How do the inputs relate to the outputs, i.e., what are the transfer and the feedback relations? If we can describe these relations analytically, describe these equations, e.g., are they linear or non-linear. State these equations, e.g., define meanings of variables and write out all relations. This algorithm is open loop. The transfer relationship is non-linear. Form the following matrix, M_{vl}, using the input data, $(t_x, t_y, t_z, \delta, \varepsilon, \zeta, f)$,

$$\mathbf{M}_{wl} = (\mathbf{P}\mathbf{T}_l\mathbf{R}_{\varepsilon,\zeta}\mathbf{T}_L)^T$$

where

$$\mathbf{T}_L = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

and

$$\mathbf{R}_{\varepsilon\zeta} = \begin{bmatrix} \cos\zeta & \sin\zeta & 0 & 0 \\ -\cos\varepsilon\sin\zeta & \cos\varepsilon\sin\zeta & \sin\varepsilon & 0 \\ \sin\varepsilon\cos\zeta & -\sin\varepsilon\cos\zeta & \cos\varepsilon & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_l = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -f \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{f} & 0 & 1 \end{bmatrix}$$

Define vectors, $\mathbf{r}_i = (m_{i1}, m_{i2}, m_{i3}, m_{i4})$, $i = 1, 2, 3, 4$, where m_{ij} is the element at the ith row and jth column of $\mathbf{M}_{wt}$. Using the input values, $(a_i, b_i, c_i, x_i^0, y_i^0, z_i^0, \alpha_i, \beta_i, \gamma_i)$ for each $i = 1, 2, \ldots, N$, form the following scalar coefficients for each of the N matching lines

$$\mathbf{A} = (x^0\mathbf{r}_1 + y^0\mathbf{r}_2) \cdot \mathbf{n}$$
$$\mathbf{B} = (x^0\mathbf{r}_2 + y^0\mathbf{r}_1) \cdot \mathbf{n}$$
$$\mathbf{C} = \mathbf{r}_1 \cdot \mathbf{n}$$
$$\mathbf{D} = \mathbf{r}_2 \cdot \mathbf{n}$$
$$\mathbf{E} = z^0\mathbf{r}_3 \cdot \mathbf{n}$$
$$\mathbf{F} = f(\alpha\mathbf{r}_1 + \beta\mathbf{r}_2) \cdot \mathbf{n}$$
$$\mathbf{G} = (\alpha\mathbf{r}_2 - \beta\mathbf{r}_1) \cdot \mathbf{n}$$
$$\mathbf{H} = \gamma\mathbf{r}_3 \cdot \mathbf{n}$$
$$\mathbf{J} = -\mathbf{r}_4 \cdot \mathbf{n}$$

Using these coefficients, form the following matrices,

$$\mathbf{A} = \begin{bmatrix} F_1 \cdots F_N & A_1 \cdots A_N \\ G_1 \cdots G_N & B_1 \cdots B_N \end{bmatrix}^T \quad (1)$$

$$\mathbf{B} = \begin{bmatrix} 0 \cdots 0 & C_1 \cdots C_N \\ 0 \cdots 0 & D_1 \cdots D_N \\ 0 \cdots 0 & -J_1 \cdots -J_N \end{bmatrix}^T \quad (2)$$

$$\mathbf{A} = [\, H_1 \cdots H_N \quad -E_1 \cdots -E_N \,]^T \quad (3)$$

Our desired output is (x, y, θ, k), so define

$$k' = \frac{1}{k}, \quad x' = k'x \quad \text{and} \quad y' = k'y \quad (4)$$

and define

$$\mathbf{q}_1 = (\cos\theta, \sin\theta) \quad \text{and} \quad \mathbf{q}_2 = (x', y', k') \quad (5)$$

Define the following matrices

$$\mathbf{D} = \mathbf{A}^T\mathbf{A} - \mathbf{A}^T\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$$

$$\mathbf{h} = \mathbf{A}^T\mathbf{C} - \mathbf{A}^T\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{C} = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$$

Define the following constants

$$c_3 = 2(a_1 + a_4)$$
$$c_2 = (a_1 + a_4)^2 + 2(a_1a_4 - a_2a_3) - (h_1^2 - h_2^2)$$
$$c_1 = c_3(a_1a_4 - a_2a_3) + 2h_1h_2(a_2 + a_3) - 2(a_1h_1^2 - a_4h_2^2)$$
$$c_0 = (a_1a_4 - a_2a_3)^2 - (h_1a_4 - h_2a_2)^2 - (h_2a_1 - h_1a_4)^2$$

Solve the following equation using these constants,

$$\lambda^4 + c_3\lambda^3 + c_2\lambda^2 + c_1\lambda + c_0 = 0 \quad (6)$$

and for each real solution, λ, find $\mathbf{q}_1$ and $\mathbf{q}_2$ that solve the following two equations:

$$\mathbf{q}_2 = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{C} - (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{A}\mathbf{q}_1$$
$$(\mathbf{D} + \lambda\mathbf{I}_2)\mathbf{q}_2 = \mathbf{h}$$

The optimal $\mathbf{q}_1$, $\mathbf{q}_2$ is taken as the pair that minimizes

$$\|\mathbf{A}\mathbf{q}_1 + \mathbf{B}\mathbf{q}_2 - \mathbf{C}\|^2 \quad (7)$$

Under what conditions are the equations over-determined or under-determined? Since a non-linear, least squares technique is used, equations should typically be overdetermined. In order for the equations not to be under-determined, there must be 3 or more non-degenerate line matches, i.e., $N \geq 3$. A line match is degenerate if (1) $(\alpha_i, \beta_i) = (0, 0)$ (i.e., the model line of a match is vertical to the ground plane), (2) for $i \neq j$, $(\alpha_i, \beta_i, \gamma_i) = (\alpha_j, \beta_j, \gamma_j)$ (i.e., the model lines are parallel) and $(a_i, b_i, c_i) = (a_j, b_j, c_j)$ (the image lines are collinear), or (3) if there is a single unique solution to the equation,

$$p_i^0 + t(\alpha_i, \beta_i, \gamma_i) = p_j^0 + t(\alpha_j, \beta_j, \gamma_j)$$

for $i \neq j$ and $p_i^0 = t(x_i^0, y_i^0, z_i^0)$ (i.e., the model lines intersect), and $(a_i, b_i, c_i) = (a_j, b_j, c_j)$ (i.e., the image lines are collinear). How do the output values vary with respect to the input, e.g., if it is non-linear, describe the nature of the non-linearity? The pose estimation depends on the solution of a fourth order polynomial with at least four real solutions. If there is increasing input noise and if the correct solution of the four suddenly no longer produces the minimum, the solution will switch to another set of the four which may, in turn, cause the output pose estimation to change suddenly in value.

Input Data Constraints: What are the constraints on the format and nature of the input? There must be three or more non-degenerate line matches, i.e., $N > 2$ (the degenerate case is already defined above).

Environmental Constraints: What are the constraints on the format and nature of the environment? (1) The lighting must be good enough to avoid excess specular reflection and shadows, since the algorithm is not explicitly designed to handle replacement errors, i.e., outliers. (2) Camera calibration must have been performed a priori. (3) There must be no roll in the camera. If there is a sensing device, what are the constraints on the type and use of the sensing (e.g., structured light, CCD camera, range camera)? CCD camera placed with the object fully within the field of view.

Knowledge Data Constraints: What are the constraints on the format and nature of the knowledge data? (1) The pitch, roll, and z-position of the part coordinate system must be identically zero in the machine coordinate system. This is the ground plane assumption. (2) Model features for matching must be linear and, furthermore, must correspond to sensed features perceivable by standard edge detection algorithms. This constrains the model to be planar polygonal.

Computing Constraints: What (if any) are the operating system requirements of the algorithm? Mathematica runs on UNIX, MacOS, Windows, Windows 95, MS DOS, Windows NT. What computer language is the source code written in? Mathematica™. Is the source code available? No (for research only). Are there any system architecture requirements for using the algorithm? No. What kind of computing hardware is required by the algorithm? Any hardware running UNIX, Macintosh, IBM-PC-compatibles. How much RAM memory and disc space is needed? About 8 megabytes RAM for Mac or PC. File size is about 2 megabytes. Are there any constraints on the numerical precision of the processing system? The computing system must allow computation that is precise to at least 24 decimal digits. This unusually high precision is due to the sensitivity of the fourth order polynomial (Eq. (6)). For a certain data set, in the presence of little or no noise, the correct solution to Eq. (6) has been found to be on the order of 10^{-21}, which would be detected as effectively zero on computing systems with less precision.

Internal Data Representation: If there is some knowledge internal to the algorithm (i.e., not input/output data) to be represented, what is the format of the representation? The object is represented in terms of the parameters for lines (not line segments) for each of the edges on the planar polygonal part.

Speed: How fast does the algorithm run, based on actual examples run on specific computers? Not measured at this time.

Complexity: What are the relations defining computational complexity of the algorithm (there may be more than one such relation) as functions of the input variables? With N equal to the number of input line matches, the algorithm complexity is roughly $335 + 153N$ time intervals. What assumptions were made in this complexity analysis? We assume that square roots, divides, adds, and multiplies are roughly equivalent in time complexity. These times can be significantly dependent on the type of computing architecture employed. Decision steps were counted as 10 time intervals and there were two of them: (1) determining which of the four solutions to the quartic (Eq. (6) are real and (2) determining the optimal $(\mathbf{q}_1, \mathbf{q}_2)$ that minimizes (Eq. (7)).

Benchmarks: If there is a standard test suite (a set of benchmarks) for algorithms performing the same type of tasks, what is that benchmark and how does the algorithm perform against standard test suite? No. Are there optimal algorithms that can produce the ideal output? No. How is optimality defined for each optimal algorithm? N/A.

Robustness: How does the algorithm perform in the presence of large perturbations (replacement noise) on a subset of its data values? It is not designed to perform successfully with replacement noise nor has it been tested under such noise. How does the algorithm perform as replacement noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? N/A. Is the algorithm able to perform well (or at all) if the input is

outside of the specified region? N/A. State all models that exist for replacement noise. None

Noise: How does the algorithm perform in the presence of small perturbations on all data values? Experimentation has been done in the presence of noise on synthetic model data, a cuboid of size $3 \times 2 \times 1.2$ m. The synthetic object was placed about 22 m from the center of the camera. The image size was (512×512) pixels. Small perturbations in translation were introduced to each ideal image line segment along its normal direction. Small perturbations in orientation were introduced by rotating each ideal image line segment with respect to its midpoint. The magnitudes of the perturbations, t and ω, of the translation and rotation, respectively, were assumed to be uniformly distributed over $[-T, T]$ pixels and $[-\Omega, \Omega]$. Monte Carlo simulations were conducted to discover the propagation of error. Tables 1 and 2 report these results. Uncertainty in these error measurements is not available. How does the algorithm perform as this type of noise varies throughout its range, e.g., does performance degrade smoothly or catastrophically? The degradation is roughly linear for the error in all dimensions as can be seen from Tables 1 and 2. State all models that exist for this type of noise. At each error level, 200 Monte Carlo simulations were done, absolute error between the ideal value of the parameter and the noisy output value was computed, and all 200 error values were averaged.

Convergence: Is the algorithm iterative or closed form? Closed form. If iterative, under what (if any) conditions is convergence guaranteed? N/A. If iterative, does the algorithm converge to the global solution? N/A

Errors: What is the error criterion, e.g., least squares? Using definitions in Eqs.(1)–(5), the task is to solve the overconstrained equation $\mathbf{A}\mathbf{q}_1 + \mathbf{B}\mathbf{q}_2 = \mathbf{C}$ for (x, y, θ, k). This is a non-linear least squares problem. The least squares solution is found by minimizing the squared residual $\|\mathbf{A}\mathbf{q}_1 + \mathbf{B}\mathbf{q}_2 - \mathbf{C}\|^2$ subject to the trigonometric constraint $\|\mathbf{q}_1\|^2 = 1$. This is accomplished by introducing a Lagrange multiplier, λ, and minimizing the following function with respect to $\mathbf{q}_1$, $\mathbf{q}_1$, and λ,

$$\varepsilon(\mathbf{q}_1, \mathbf{q}_2, \lambda) = \|\mathbf{A}\mathbf{q}_1 + \mathbf{B}\mathbf{q}_2 - \mathbf{C}\|^2 + \lambda(\|\mathbf{q}_1\|^2 - 1).$$

What input errors and/or internal errors does the algorithm detect? None. What does it do if such errors

Table 1. Propagation of error: Image feature translation error versus pose and scale error (direction error fixed at three degrees, number of line matches fixed at ten).

Number of pixels (translation error)	2	4	6	8	10	12	14	16
Pose error (x only) in meters	0.08	0.16	0.24	0.32	0.40	0.48	0.56	0.64
Pose error (y only) in meters	0.04	0.09	0.14	0.19	0.24	0.29	0.34	0.39
Pose error (only) in degrees	0.5	0.54	0.58	0.62	0.66	0.7	0.74	0.78
Scale error	0.03	0.045	0.06	0.075	0.09	0.105	0.13	0.145

Table 2. Propagation of error: Image feature direction error versus pose and scale error (translation error fixed at three pixels, number of line matches fixed at ten).

Number of degrees (direction error)	2	4	6	8	10	12	14	16
Pose error (x only) in meters	0.08	0.16	0.24	0.32	0.40	0.48	0.56	0.64
Pose error (y only) in meters	0.04	0.09	0.14	0.19	0.24	0.29	0.34	0.39
Pose error (only) in degrees	0.1	0.4	0.7	1.0	1.3	1.6	1.9	2.2
Scale error	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10

are detected? N/A Are there error recovery procedures? No.

Reliability: If the algorithm is available as source code, a library, or embedded in a system, how reliable is it? Not known. Are there known bugs? No. What reliability tests has the algorithm passed? None. Has there been testing by some sort of coverage tool, which keeps track of which code is executed in a given session? No.

Testing and Analysis: What experiments have been done with the algorithm? Experimentation has been done in the presence of noise on synthetic model data, a cuboid of size $3 \times 2 \times 1.2$ m. The synthetic object was placed about 22 m from the center of the camera. Image size was (512×512) pixels. If input data is varied over a set of variables, what criteria are chosen to sample the space of variables in order to generate sample input data? Small perturbations in translation and direction were introduced as described earlier. Was Monte Carlo testing done? Monte Carlo simulations were conducted to discover the propagation of error. Tables 1 and 2 report these results. Are there simulators available to generate input data? Yes, but only for the Mathematica code.

What kinds of analysis have been done on the results? Propagation of error analysis using Monte Carlo simulations of small perturbations on the sensed input lines. What kinds of graphs and tables have been produced and what is their format? See Tables 1 and 2. What statistical methods have been used? Simple arithmetic mean on all the errors as shown in Tables 1 and 2.

Application Experience: How widely has the algorithm been used? Not known. What is the reported experience with the algorithm? None.

Upstream and Downstream Requirements: What kinds of components or algorithms are typically executed prior to or subsequent to this algorithm? Matching algorithms are typically upstream. Are the representations chosen for the input and output data consistent with the expected or typical upstream and downstream algorithms? The expected upstream algorithm is one matching a sensed feature to model feature and, since the Tan algorithm requires input line matches for edges on a planar polygonal object, the matching algorithm must produce line matches as well. However, if only point or line segment matches are available from the matching algorithm, line parameters can easily be generated from them. Does anything in the algorithm constrain the upstream or downstream algorithms that must be used? Line matches (or matches from which line matches can be easily derived) must be presented to the algorithm. Surface or curve matches are not acceptable.

Parallelizability: Can the algorithm be parallelized? Some minor aspects of the algorithm can be parallelized. Has it been? No. How does parallelizing affect performance? Very little.

Modularizability: Can subcomponents of the algorithm be modularized? Yes. Have they been? Yes. How does modularization affect performance? Slows.

Nature of Interaction: How does the component or algorithm interact with other algorithms and systems? In Mathematica™, it is a simple function call. In order for Mathematica to exchange data with other software and systems, Mathematica's MathLink™ communications protocol must be used.

Coding Style: How is the code written, e.g., in functional, procedural, recursive, or object-oriented style? Mathematica code is functional, procedural, and interpreted.

Compliance to Standards: Which (if any) interface or data standards (published or de facto) are used and complied with? None.

3.3. Step 3: Formal Language Instantiation of the Specification

In order to study unambiguous, semantically-rich representations of components, a goal of this research was to use a formal information modeling language to record information about particular components in the specification categories. However, the goal of the research was not to find the best formal language for the representation of components. We have chosen the EXPRESS language to conduct a feasibility study on formally representing the components. EXPRESS was developed as an information modeling language and is part of the ISO standard 10303, also known as Standard for the Exchange of Product Model Data (STEP) (ISO, 1994a). The authors examined formal languages, such as Z (Diller, 1997), for modeling components. Although Z and other similar languages provide sufficient richness for expressing the component specifications, they do not provide some easily-used capabilities of EXPRESS. Team members were also already familiar with EXPRESS.

The EXPRESS language proved to be adequate for representing the generic and instantiated component specifications. Using the generic model of the component family in EXPRESS, particular components may be defined in STEP Part 21 exchange files (ISO, 1994b). Part 21 files can be used in conjunction with available tools to read them into computer programs or to generate them automatically. STEP Part 21 files are difficult to read by humans unfamiliar with their syntax. We developed a computer-aided software engineering tool to aid in the definition of the part pose estimation specifications. The tool, named FFProbe, was built using a NIST-developed Data Probe (Morris, 1993) system, which allows users to build an interactive graphical system for dealing with instances of data that correspond to a particular EXPRESS schema. Using FF-Probe, users can create or edit instances of component specifications in a readable format, save collections of instances in a STEP Part 21 file, read in collections of instances from STEP Part 21 files, and examine the models in a readable way on a computer screen. The name FFProbe refers to "functionality frames," which

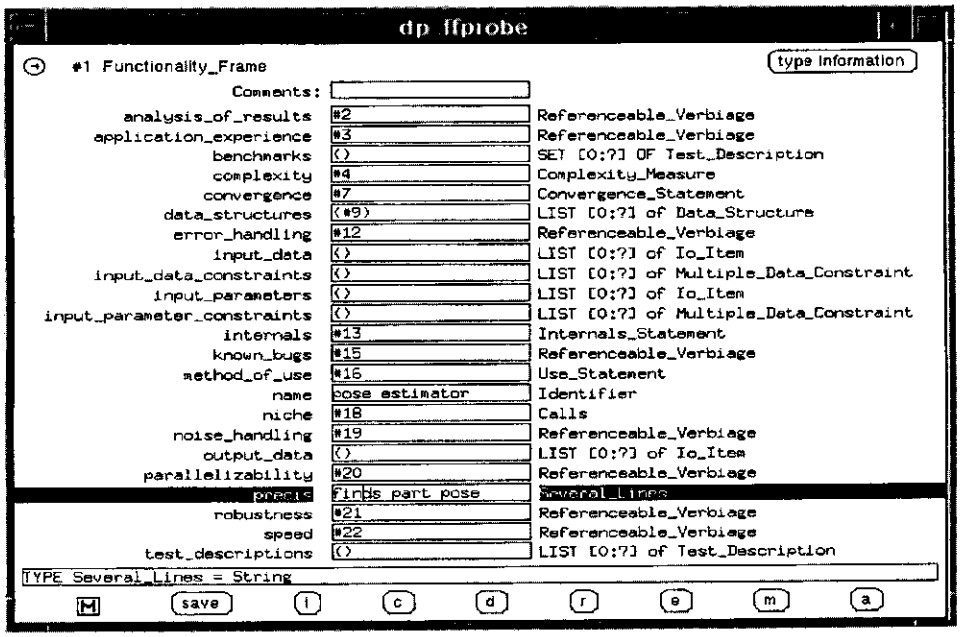


Figure 2. FFProbe functionality frame window.

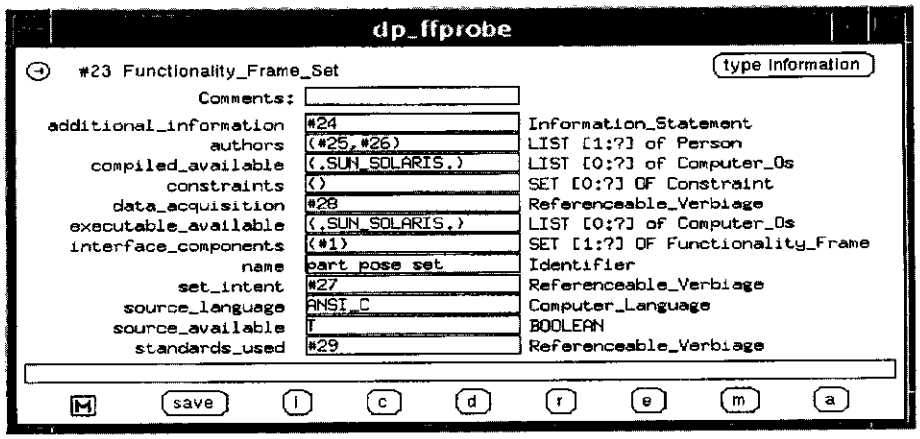


Figure 3. FFProbe functionality frame set window.

```
DATA;
#1=FUNCTIONALITY_FRAME(#2, #3, (), #4, #7,(#9), #12,
     (), (), (), (), #13, #15, #16, 'pose estimator', #18,
     #19,(), #20, 'finds part pose', #21, #22, ());
#2=REFERENCEABLE_VERBIAGE(('uses Monte Carlo', 'error tables tabulated'));
#3=REFERENCEABLE_VERBIAGE(('none'));
#4=COMPLEXITY_MEASURE(#5, #6);
#5=REFERENCEABLE_VERBIAGE(('order log N'));
#6=REFERENCEABLE_VERBIAGE(('quite efficient'));
#7=CONVERGENCE_STATEMENT(#8, .T.);
#8=REFERENCEABLE_VERBIAGE(('fast convergence'));
#9=DATA_STRUCTURE('points',#11, #10, .T.);
#10=REFERENCEABLE_VERBIAGE(('double points[20];'));
#11=REFERENCEABLE_VERBIAGE(('array of twenty doubles'));
#12=REFERENCEABLE_VERBIAGE(('aborts on error'));
#13=INTERNALS_STATEMENT((), #14,());
#14=REFERENCEABLE_VERBIAGE(('internals not available'));
#15=REFERENCEABLE_VERBIAGE(('no known bugs'));
#16=USE_STATEMENT(#17, .CALL.);
#17=REFERENCEABLE_VERBIAGE(('used by function call'));
#18=CALLS((), ());
#19=REFERENCEABLE_VERBIAGE(('chokes on noFigure 1 - zooise'));
#20=REFERENCEABLE_VERBIAGE(('must be sequential'));
#21=REFERENCEABLE_VERBIAGE(('very robust'));
#22=REFERENCEABLE_VERBIAGE(('20 seconds on SPARC for 300 points'));
#23=FUNCTIONALITY_FRAME_SET(#24, (#25, #26), (.SUN_SOLARIS.), (), #28,
     (.SUN_SOLARIS.), (#1), 'part pose set', #27, .ANSI_C., .T., #29);
#24=INFORMATION_STATEMENT((),(#25, #26), ());
#25=PERSON('Jones', 'Pete', 'USA', '800-123-4567', 'pete@uw.edu',
     '', '');
#26=PERSON('Nimble', 'Jack', 'USA', '800-123-4568', 'jack@uw.edu',
     '', '');
#27=REFERENCEABLE_VERBIAGE(('find part pose'));
#28=REFERENCEABLE_VERBIAGE(('camera pixels'));
#29=REFERENCEABLE_VERBIAGE(('written in ANSI C'));
ENDSEC;
```

Figure 4. Data section of file written by FFProbe.

is another term used to describe component specification instantiations.

Figure 2 shows a window displayed by the FFProbe for editing a functionality frame. The window has been filled in with data for a fictitious part pose estimator, in order to present a more compact example. The EXPRESS model for the generic specifications described in section 3.1 was 15 pages long. The names of the attributes are shown on the left, the data types on the right, and the data values in the middle. Most of the data values are references (such as #2) to other data instances. The names and data types of the attributes are taken from the EXPRESS schema for the specification. Figure 3 shows a FFProbe window for editing a functionality frame set, also filled in with fictitious data. All the referenced data instances of Figs. 2 and 3 are shown in Fig. 4, which is the data section of the STEP Part 21 file written by the FFProbe after all the required data needed to complete the example was entered. In Fig. 4, data types which are strings appear between single quotes, data types that are lists appear inside parentheses, and data types that are EXPRESS entity instances appear as references of the form #n. The functionality frame shown in Fig. 2 is item #1 in

Fig. 4. The functionality frame set shown in Fig. 3 is item #23 in Fig. 4.

Once functionality frame data has been put into STEP Part 21 files, readily available software tools automatically generate C++ code for accessing the data. These access functions could be used in a computer program for searching a database of software component specifications. Database systems already exist that enable some forms of searching automatically, given an EXPRESS schema and data conforming to the schema.

4. Future Work

We are performing further validation of the component specifications for part pose estimation algorithms. Other researchers at NIST, who are developing and testing new part pose estimation algorithms, will use our component specification framework. This will verify the approach and categories we have selected and will expand the library of available part pose components for which we have specifications. Since our approach relies on framework-dependent specifications, we are investigating formal specification languages that go beyond a single component. We are starting to experiment with Rapide, an Architectural Description Language developed at Stanford University (Luckham et al., 1995). Architectural Description Languages rigorously describe the design of software systems and are intended to facilitate better understanding of software architectures and facilitate reuse. Within this context, we are going to formally describe the RCS architecture and specify its constituent nodes in a generic fashion. Specific instantiations of the nodes or their internal modules (BG, WM, SP, VJ) will be created for particular example applications. We will study the categorization of component and algorithm families that are relevant to robotics and intelligent systems. The resulting, formally-defined framework will allow researchers to experiment with the sharing and reusability of intelligent control components.

5. Conclusions

We have examined the feasibility of expressing a software component's specifications in three stages: the generic specification categories for the family of components, the natural language instantiations for particular components within that family, and the formal language instantiations for the components. Our results indicate that this is a viable approach to facilitate the communication the relevant facets of a component to potential users. Given agreed-upon, common templates for specific component categories, this approach can be adopted by the robotics research community for exchanging algorithms and software. Having a common set of domain-specific functional categories, it is possible to generate major component families that can be shared. It is not essential that the research community adopt formal representations for their components. Since researchers in the same field tend to have a common vocabulary and terminology already, the natural language versions should be an adequate and necessary first step for publication and retrieval of components. As the specification categories are exercised and refined through use, formalization may become appropriate to consider. At that time, it will be possible to consider tools that relieve users of the burden of viewing the formal specifications. Internet-based component search, matching, retrieval, simulation, and composition will eventually be possible. The initial step—agreeing upon component families and specification categories for robotics—has to be taken first.

References

Albus, J. 1996. An engineering architecture for intelligent systems. In *Proceedings of the American Association for Artificial Intelligence (AAAI), Fall Symposium Series.*

Albus, J.S. 1997. 4-D/RCS: A reference model architecture demo III, Technical Report NIST Technical Note 5994, U.S. Department of Commerce, National Institute of Standards and Technology, NIST, Gaithersburg, MD 20899.

Bailo, C., Alderson, G., and Yen, C. 1994. Requirements of open, modular, architecture controllers for applications in the automotive industry, white paper. Available on the Wide World Web http://www.arcweb.com/omac/Documents/omacv11.htm.

Baresi, L., Orso, A., and Pezzè, M. 1997. Introducing formal specification methods in industrial practice. In *Proceedings of the 1997 International Conference on Software Engineering*, pp. 56–66.

Basili, V. and Abd-El-Hafiz, S. 1992. Packaging reusable components: The specification of programs, Technical Report CS-TR-2957, UMIACS-TR-92-97, Department of Computer Science, University of Maryland.

Batory, D. and O'Malley, S. 1992. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering Methodology*, 1(4):355–398.

Berners-Lee, T. and Connolly, D. 1993. Hypertext markup language: A representation of textual information and metainformation for retrieval and interchange, Technical Report, CERN and Atrium Technology Inc.

Brockschmidt, K. 1995. *Inside OLE*, 2nd edition, Microsoft Press: Redmond, WA.

Chappell, D. 1996. *Understanding ActiveX and OLE—A Guide for Developers and Managers*. Microsoft Press: Redmond, WA.

Chen, Y. and Cheng, B. 1997. Formally specifying and analyzing architectural and functional properties of components for reuse. In *WISR8, the 8th Workshop on Software Reuse*. http://www.umcs.maine.edu/ftp/wisr/wisr8/papers.html.

Dementhon, D. and Davis, L. 1995. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141.

Demeyer, S., Meijler, T., Nierstrasz, O., and Steyaert, P. 1997. Design guidelines for 'tailorable' frameworks. *Communications of the ACM*, 40(10).

Denning, A. 1997. *ActiveX Controls Inside Out*. Microsoft Press: Redmond, WA.

Diller, A. 1997. *Z: An Introduction to Formal Methods*, 2nd edition, John Wiley & Sons: New York, NY.

Fayad, M. and Schmidt, D.C. 1997. Object-oriented application frameworks—Introduction. *Communications of the ACM*, 40(10): 32–38.

Haralick, R.M. 1992. Performance characterization in computer vision. In *Proceedings of the 3rd British Machine Vision Conference*.

Haralick, R.M. et al. 1989. Pose estimation from corresponding point data. In *Machine Vision for Inspection and Measurement*.

Herman, M. and Albus, J. 1988. Overview of the multiple autonomous underwater vehicles (MAUV) project. In *IEEE International Conference on Robotics and Automation*, Philadelphia, PA.

Huang, H. 1996. An architecture and a methodology for intelligent control. *IEEE Expert*, 11(2).

Huttenlocher, D.P. and Ullman, S. 1990. Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5(2):195–212.

IEEE 1994. IEEE Std 1076-1993: IEEE standard VHDL language reference manual, IEEE Standards.

ISO 1994a. Industrial automation systems and integration, product data representation and exchange—part 11. EXPRESS Language Reference Manual. International Organization for Standardization.

ISO 1994b. Industrial automation systems and integration, Product data representation and exchange—part 21: Clear text encoding of the exchange structure, International Organization for Standardization.

JavaSoft 1998. Products and APIs. See Web URL: http://www.java.sun.com/products.

Kara, D. 1997. Seeing the forest for the trees. *Software Magazine*.

Kazman, R., Clements, P., Bass, L., and Abouwd, G. 1997. Classifying architectural elements as a foundation for mechanism matching. In *Proceedings of the 1997 21st Annual International Computer Software and Applications Conference*.

Krueger, C. 1992. Software Reuse. *ACM Computing Surveys*, 24(2).

Liskov, B. and Wing, J. 1993. Specifications and their use in defining subtypes. In *Proceedings of the 8th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 16–28.

Luckham, D.C., Kenney, J.J., Augustin, L.M., Vera, J., Bryan, D., and Mann, W. 1995. Specification and analysis of system architecture using rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355.

McCool, R. 1994. The common gateway interface. NCSA, 1.1 edition.

Morris, K. 1993. Data probe: A tool for EXPRESS-based Data. In *Proceedings of the 7th Annual Engineering Database Symposium—ASME Computers in Engineering Conference*.

Murphy, K., Juberts, M., Legowik, S., Nashman, M., Schneiderman, H., Scott, H., and Szabo, S. 1993. Ground vehicle control at NIST: From teleoperation to autonomy. In *Proceedings of the 7th Annual Space Operations, Applications, and Research Symposium*, Houston, TX.

Murphy, K., Norcross, R., and Proctor, F. 1988. CAD directed robotic deburring. In *Proceedings of the Second International Symposium on Robotics and Manufacturing Research, Education, and Applications*, Albuquerque, NM.

Musser, D.R. and Saini, A. 1996. *STL Tutorial and Reference Guide*, Addison Wesley: Reading, MA.

Nauer, P. and Randel, E.B. 1968. *Software Engineering: Report on a Conference by the NATO Science Committee*, NATO Scientific Affairs Division: Brussels, Belgium.

Nierstrasz, O., Gibbs, S., and Tsichritzis, D. 1992. Component-oriented software development. *Communications of the ACM*, 35(9).

Nierstrasz, O. and Meijler, T. 1995. Research directions in software composition. *ACM Computing Surveys*, 27(2).

OMAC API Workgroup 1997. OMAC API Set. Open Modular Architecture Controls (OMAC) User Group. See Web URL: http://isd.cme.nist.gov/info/omacapi.

OMG 1996. Description of the New OMA Reference Model, Draft 1. OMG Document ab/96-05-02. Object Management Group, Framingham, MA. http://www.omg.org.

OMG 1997. The common object request broker: Architecture and specification. Object Management Group Formal Document 97-02-25. Object Management Group, Framingham, MA. http://www.omg.org.

OSACA 1998. Open system architecture for controls within automation systems. See Web URL: http://www.osaca.org.

Paredis, C., Brown, B., and Khosla, P. 1997. A rapidly deployable manipulator system. *Robotics and Autonomous Systems*, 21:289–304.

Piccinocchi, S., Ceccarelli, M., Piloni, F., and Bicchi, A. 1997. Interactive benchmark for planning algorithms on the web. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*.

Podgurski, A. and Pierce, L. 1993. Retrieving reusable software by sampling behavior. *ACM Transactions on Software Engineering and Methodology*, 2(3):286–303.

Postel, J. (Ed.). 1981. Transmission control protocol—DARPA internet program protocol specification. RFC 793.

Pree, W. 1997. Component-based software development—A new paradigm in software engineering? *Software-Concepts and Tools*, 18:169–174.

Rogerson, D. 1997. *Inside COM*, Microsoft Press: Redmond, WA.

Short, K. 1997. Component based development and object modeling. See Web URL: http://www.cool.sterling/cbd/whitepaper.

SML Corporation 1998. Open system environment for controller (OSEC) architecture overview. Published on the Wide World Web http://www.sml.co.jp/OSEC.

Steenstrup, M., Arbib, M.A., and Manes, E.G. 1983. Port automata and the algebra of concurrent processes. *Journal of Computer and System Sciences*, 27(1):29–50.

Stewart, D., Volpe, R., and Khosla, P. 1997. Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Transactions on Software Engineering*, 23(12):169–174.

Stouffer, K. and Russell, R. 1995. ADACS—An advanced deburring and chamfering system. In *Proceedings of the 6th International Conference on Manufacturing Engineering*, Melbourne, Australia.

Szyperski, C. 1998. *Component Software: Beyond Object-Oriented Programming*. Addison Wesley Longman Limited: Essex, England.

Tan, T., Sullivan, G., and Baker, K. 1992. Linear algorithms for object pose estimation. In *Proceedings of the 3rd British Machine Vision Conference*.

Tan, T., Sullivan, G., and Baker, K. 1994, Pose determination and recognition of vehicles in traffic scenes. In *European Conference on Computer Vision*.

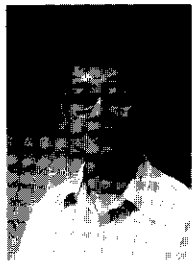
Tan, T., Sullivan, G., and Baker, K. 1996. Pose determination and recognition of vehicles in traffic scenes. *Pattern Recognition*, 29(3):449–461.

Zaremski, A. and Wing, J. 1996. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):449–461.

Elena Messina is Leader of the Knowledge Systems Group in the Intelligent Systems Division at the National Institute of Standards and Technology. Her research interests include world modeling, knowledge representation, and software engineering methodologies for developing intelligent systems. Prior to joining NIST, she worked in robotics at Cincinnati Milacron and in Computer-Aided Design and Manufacturing at the Structural Dynamics Research Corporation. She received a B.S. in Engineering Science from the University of Cincinnati in 1980.

John Horst has been with the Intelligent Systems Division of the National Institute of Standards and Technology since 1988. He has B.A., BSEE, and MSEE from the University of Maryland, College Park. His current areas of interest are control systems, computer vision, and component-based design for manufacturing systems. He is currently pursuing a Ph.D. in Electrical Engineering at the University of Maryland.

Thomas Kramer has been a Guest Researcher at NIST and a Research Associate of the Catholic University of America since 1984. He received a B.A. in physics from Swarthmore College in 1965, followed by two years in the Peace Corps. He received his Ph.D. in mathematics from Duke University in 1971, after which he was a science and technology bureaucrat for 13 years. His research interests include automated reasoning, feature-based manufacturing, other aspects of automated manufacturing, and formal languages.



Hui-Min Huang is currently a mechanical engineer with the National Institute of Standards and Technology (NIST), U.S. Department of Commerce. His major research areas include the architectures and software engineering methodologies for real-time, intelligent control systems. His prior professional experiences include technical positions with the Singer Company Link Simulation Division (Silver Spring, MD) and the Timex Watches Company (Taiwan). Mr. Huang received an M.S. degree in Mechanical Engineering from the University of Texas at Arlington in 1982 and an M.S. degree in Computer Science from the Johns Hopkins University in 1986.

John Michaloski has been a computer scientist at NIST for over 15 years. His research deals with open-architecture controllers, advanced robotics, and automation technology for manufacturing systems. He earned his M.S. in computer science from the Georgia Institute of Technology, and his B.S. in mathematics from the University of Maryland.